



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
-----------------	-------------	----------------------	---------------------	------------------

10/791,602

03/02/2004

Carl A. Waldspurger

A43

2029

69355 7590 08/02/2007  
VMWARE / JEFFREY PEARCE  
DARRYL SMITH  
3401 Hillview Ave.  
PALO ALTO, CA 94304

EXAMINER

TURCHEN, JAMES R

ART UNIT

PAPER NUMBER

2139

MAIL DATE

DELIVERY MODE

08/02/2007

PAPER

**Please find below and/or attached an Office communication concerning this application or proceeding.**

The time period for reply, if any, is set in the attached communication.

## Office Action Summary

Application No.

10/791,602

Applicant(s)

WALDSPURGER ET AL.

Examiner

James Turchen

Art Unit

2139

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

### Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

### Status

- 1) ☒ Responsive to communication(s) filed on 20 March 2004.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

### Disposition of Claims

- 4) ☒ Claim(s) 1-47 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-47 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

### Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 03/20/2004 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

### Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- \* See the attached detailed Office action for a list of the certified copies not received.

### Attachment(s)

- ☒ Notice of References Cited (PTO-892)
- ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- ☐ Information Disclosure Statement(s) (PTO/SB/08)  
Paper No(s)/Mail Date \_\_\_\_\_
- ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_
- ☐ Notice of Informal Patent Application
- ☐ Other: \_\_\_\_\_

### **DETAILED ACTION**

Claims 1-47 are pending.

#### ***Claim Rejections - 35 USC § 101***

Claim 1 is rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. It is unclear whether the claim pertains to a system or a method.

#### ***Claim Rejections - 35 USC § 103***

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

Claim 2 is rejected under 35 U.S.C. 103(a) as being unpatentable over Nachenberg (US 6,021,510, hereinafter '510) in view of Nachenberg (US 5,826,013, hereinafter '013).

Regarding claims 2 and 35:

'510 discloses a computer that includes at least a processor that executes instructions stored on a memory (column 1 lines 60-63, a computer inherently has a processor that executes instructions), which is organized into separately addressable memory blocks (column 3 line 26, the file is divided into sectors; examiner interprets memory block to be an individually addressable unit), a method for verifying the validity of instructions comprising:

- determining an identifying value for a current memory block that contains the current instruction (column 4 lines 40-47, module 5 computes the hash values for each of the previously scanned sectors)

- comparing the identifying value of the current memory block with a set of reference values (column 4 lines 40-47, module 5 compares the new hash value with a previously recorded hash value)

- if the identifying value satisfies a validation condition, allowing execution of the current instruction by the processor (column 4 lines 48-53, after determining that the hash values are identical, the file is considered "unchanged"; it is inherent that a file be allowed to execute after being deemed virus-free).

- If the identifying value does not satisfy the validation condition, generating a response (column 4 lines 54-63, module 5 generates a virus scan and notifies the user).

'510 does not explicitly state that the current instruction is verified dynamically before being executed. '013 discloses examining the instruction/interrupt usage profiles of each known polymorphic virus as each instruction is fetched for emulation (column 3

Art Unit: 2139

lines 36-40). It would have been obvious to one of ordinary skill in the art at the time of invention to modify the method of '510 with the examining of instructions after being fetched in order to flag polymorphic viruses that deploy mutation engines (column 3 lines 36-46).

Regarding claims 3, 34, and 36:

Nachenberg ('510 and '013) discloses the method of claim 2, further comprising including in the set of reference values at least one validation entry corresponding to at least one identifying value for predetermined contents of a known, valid memory block ('510; column 4 lines 48-53, the old hash value (predetermined contents) corresponds to the new hash value (validation entry)) in which the validation condition is that the identifying value of the current memory block matches any validation entry in the set of reference values (the new hash value will match the old hash value that is within the set of old hashes, each hash pertaining to a sector).

Regarding claims 4 and 37:

Nachenberg discloses the method of claim 2, further comprising including in the set of reference values at least one invalidation entry corresponding to at least one identifying value for predetermined contents of a known invalid memory block in which the validation condition is that the identifying value of the current instruction differs from all invalidation entries in the set of reference values ('013; column 6 lines 54-63, virus profile data is checked in order to identify a known virus).

Regarding claims 5 and 38:

Nachenberg discloses the method of claim 2, in which the step of determining the identifying value of the current-memory block comprises computing a hash value as a function of at least a sub-set of the contents of the current memory block and each reference value is computed as a hash value of at least a sub-set of a known, reference memory block ('510; column 3 line 55-column 4 line 63, each sector (memory block) has a computed hash value and is checked with a previously computed hash value to check for modifications).

Regarding claims 6 and 39:

Nachenberg discloses the method of claim 5, in which the step of computing the hash value of both the current memory block and at least one of the reference memory blocks comprises computing the respective hash values based on only part of the contents of both the current and the reference memory blocks ('510; column 5 lines 6-33, the method only scans those sectors actually retrieved by module 5).

Regarding claims 7 and 40:

Nachenberg discloses the method of claim 6, further comprising identifying, in at least one reference memory block, non-indicative contents that are valid but that are at least potentially non-constant such that they do not indicate validity of the reference memory block ('510; column 5 lines 6-33, the method only scans those sectors actually retrieved by module 5); before or when computing the hash value for the reference memory block, applying a mask to the contents of the reference memory block such that the non-indicative contents do not influence the computed hash value ('510; column 5 lines 6-33, the method only scans those sectors actually retrieved by module 5, the non-

Art Unit: 2139

indicative components do not influence the computed hash value); and before or when computing the hash value for the current memory block, applying the mask to the current memory block contents. Although Nachenberg does not explicitly state how the individual sectors are selected and scanned, masks are well known in the art and it would have been obvious to one of ordinary skill in the art at the time of invention to apply a mask in order to extract those sectors that are to be scanned.

Regarding claims 8-10 and 41-43:

Nachenberg discloses the method of claim 2, further comprising for each of a plurality of memory blocks, indicating in a structure whether each respective block is validated ('510; column 3 lines 46-54, register means includes hardware, software, and/or firmware registers, stacks, flags, automata, indication bits, etc; a stack is a structure) and for each current instruction from a memory block whose structure indication is that it is validated, directly allowing execution of the current instruction (it is inherent that if a file is virus-free, then to allow execution of the current instruction).

Regarding claims 11 and 44:

Nachenberg discloses the method of claim 8, in which the step of indicating in the structure whether the plurality of memory blocks is validated comprises making a corresponding entry in a software data structure (('510; column 3 lines 46-54, register means includes hardware, software, and/or firmware registers, stacks, flags, automata, indication bits, etc; a stack is a structure).

Regarding claim 12:

Nachenberg discloses the method of claim 8, further comprising performing the steps of determining the identifying value for the current memory block and comparing the identifying value of the current memory block with a set of reference values only for current instructions located in memory blocks not indicated in the structure as being validated and if the identifying value of a current memory block not indicated as being validated satisfies the validation condition, then setting the corresponding structure indication to indicate that it is validated ('510; column 3 line 58-column 4 line 64, the current sector matches a validation condition (matches the reference value) then it rescans the file for viruses, it is inherent to mark the file as virus-free or contaminated).

Regarding claim 13:

Nachenberg discloses the method of claim 12, further comprising sensing modification of any memory block for which the structure includes an indication and, upon sensing modification of any such memory block, setting its indication in the structure to indicate that the memory block is not validated (column 4 lines 31-39, the file is checked for a change in size and scanned if the size has changed).

Regarding claim 14:

Nachenberg discloses the method of claim 8, further comprising determining a branch history for the current instruction and checking whether the memory blocks in which instructions in the branch history are located are validated, the validation condition including the requirement that each checked memory block in the branch history is validated (it is inherent for computer systems to have a method of branch prediction and to predict the future outcome based on previous occurrences).



Regarding claim 15:

Nahcenberg discloses the method of claim 2, further comprising performing the steps of determining the identifying values for current memory blocks, comparing the identifying values with the set of reference values, and determining whether the validation condition has been satisfied only after the occurrence of a triggering event (it is inherent that a file be checked after the occurrence of a triggering event whether by a user, a time-based trigger, an active trigger (upon program load), etc).

Regarding claim 16:

Nahcenberg discloses the method of claim 15, in which the triggering event is the writing of at least one new unit of code or data ('510; column 3 lines 35-36, the file changed thus causing a new unit of data).

Regarding claim 17:

Nachenberg discloses the method of claim 15, but does not disclose in which the triggering event is the attempted execution of any instruction located on any memory. It is well known in the art at the time of invention to dynamically scan a program as it is being "opened" or "executed". Thus, it would have been obvious to one of ordinary skill in the art at the time of invention to modify the method of Nachenberg to scan a program in order to ensure the integrity of an executable program while it is running or to detect infections between a program's integrity check and execution (A Generic Virus Scanner in C++, page 6, section 2.5).

Regarding claim 18:

Nachenberg discloses the method of claim 15, in which the triggering event is the attempted execution of any instruction located on any unverified memory block of newly installed software ('510; column 3 lines 26-40, the newly installed software would be scanned based on the fact that it is being examined for the first time)

Regarding claims 19-21:

Nachenberg discloses the method of claim 15, further comprising triggering dynamic verification depending on the identity of the user of the computer who has caused submission of the current instruction (it is inherent that a computer has a method for access control based on the user of the system in that the program will not load if the user does not have permission to run it).

Regarding claim 22:

Nachenberg discloses the method of claim 2, further comprising verifying only a sample of the current instructions ('510; figure 1 shows the file is divided into a plurality of sectors and only sectors 1, 2, 3, and J are scanned).

Regarding claim 23:

Nachenberg discloses the method of claim 22, in which the sample is a time-sampled sub-set of current instructions (it is inherent that the virus scanner disclosed by Nachenberg depends upon some period of time elapsing between samples).

Regarding claim 24:

Nachenberg discloses the method of claim 22, in which the sample is a sequentially sampled sub-set of current instructions ('510; figure 1 shows the sectors 1, 2, and 3; it is inherent that these sectors would be sequentially sampled).

Art Unit: 2139

Regarding claim 25:

Nachenberg discloses the method of claim 22, in which the sample is a sub-set of current instructions sampled spatially, over a range of addresses or equivalent memory block identifiers ('510, figure 1 shows sectors 1, 2, 3, etc. these sectors are a set size and spatially sampled).

Regarding claims 26 and 27:

Nachenberg discloses the method of claim 2, but does not disclose in which the step of generating the response comprises terminating a software entity with which the current memory block is associated, however, it was well known in the art at the time of invention to suspend or cancel a current software's execution when a virus has been detected. It would have been obvious to one of ordinary skill in the art at the time of invention to cancel or suspend the infected software in order to prevent the virus from activating and to allow disinfection of the virus.

Regarding claim 28:

Nachenberg discloses the method of claim 2, in which the step of generating the response comprises posting a message to a user, system administrator, or other predetermined recipient ('510; column 4 lines 48-63, sends a message to the user to via the interface).

Regarding claim 30:

Nachenberg discloses in a computer that includes a virtual machine running in a direct execution mode on an underlying hardware platform via an intermediate software layer ('013, column 3 lines 6-16, the CPU emulator is a virtual machine that tricks the

Art Unit: 2139

virus into thinking it is being run on the CPU) and the method as in claim 2, in which the step of generating the response includes checkpointing the state of the virtual machine ('013, column 12 lines 38-43, if a match is detected, the module marks a register and moves on to the next instruction).

Regarding claim 31:

Nachenberg discloses the method of claim 2, further comprising associating different responses with at least two different memory blocks and upon detection of failure of the current instruction to satisfy the validation condition, generating the response associated with the memory blocks in which the current instruction is located ('510, column 4 lines 48-53, the determination is sent to the user, it is inherent that the determination will be different for different memory blocks).

Regarding claim 32:

Nachenberg discloses the method of claim 2, further comprising tracking which programs are being executed within the computer by associating the reference values with respective predetermined programs, a match between the identifying value of the current memory block with any validation entry in the set of reference values indicating execution of the corresponding one of the predetermined programs ('510, column 1 lines 27-36, a hash value is associated with a program/file).

Claims 29 and 33 rejected under 35 U.S.C. 103(a) as being unpatentable over Nachenberg as applied to claim 2 above, and further in view of Complete Computer System Simulation: The SimOS Approach (hereinafter SimOS).

Regarding claims 29, 33, and 45-47:

Nachenberg discloses in a computer that includes a virtual machine running in a direct execution mode on an underlying hardware platform via an intermediate software layer ('013, column 3 lines 6-16, the CPU emulator is a virtual machine that tricks the virus into thinking it is being run on the CPU) and the method as in claim 2, but does not disclose the step of generating a response comprises switching the execution mode of the virtual machine to binary translation. SimOS discloses the ability to switch between direct execution and binary translation modes (page 40, Switching simulators and sampling). It would have been obvious to one of ordinary skill in the art at the time of invention to modify the method of Nachenberg to allow switching of modes in order to dynamically generate code supports on-the-fly changes of the simulator's level of detail (page 39, 2<sup>nd</sup> column, 3<sup>rd</sup> paragraph).

### ***Conclusion***


Any inquiry concerning this communication or earlier communications from the examiner should be directed to James Turchen whose telephone number is 571-270-1378. The examiner can normally be reached on MTWRF 7:30-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ayaz Sheikh can be reached on (571)272-3795. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Art Unit: 2139

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

JRT



CHRISTIAN L. HOGGINS  
AU 2131